# fauxmo Documentation

*Release 0.3.6*

**Nathan Henrie**

**Apr 08, 2017**

# Contents

Contents:

CHAPTER 1

# fauxmo package

## Subpackages

### fauxmo.handlers package

#### Submodules

#### fauxmo.handlers.hass module

**class** fauxmo.handlers.hass.**HassAPIHandler**(*host*, *password*, *entity*, *port=8123*)
Bases: object

Handler for Home Assistant (hass) Python API.

Allows users to specify Home Assistant services in their config.json and toggle these with the Echo. While this can be done with Home Assistant's REST API as well (example included), I find it easier to use the Python API.

**off**()

**on**()

**send**(*signal*)
Send a signal to the hass *call_service* function, returns True.

The hass Python API doesn't appear to return anything with this function, but will raise an exception if things didn't seem to work, so I have it set to just return True, hoping for an exception if there was a problem.

**Args:** signal (const): signal imported from homeassistant.const. I have imported SERVICE_TURN_ON and SERVICE_TURN_OFF, make sure you import any others that you need.

### fauxmo.handlers.rest module

**class** fauxmo.handlers.rest.**RESTAPIHandler**(*on_cmd*, *off_cmd*, *method='GET'*, *on_data=None*, *off_data=None*, *on_json=None*, *off_json=None*, *headers=None*, *auth_type=None*, *user=None*, *password=None*)

Bases: object

Rest API handler class.

The Fauxmo class expects handlers to be instances of objects that have on() and off() methods that return True on success and False otherwise. This class takes a mix of url, method, header, body, and auth data and makes REST calls to a device.

**off**()

**on**()

**set_state**(*cmd*, *data*, *json*)
Call HTTP method, for use by *functools.partialmethod*.

### Module contents

## Submodules

## fauxmo.cli module

cli.py Argparse based CLI for fauxmo. Helps provide console_script via argparse. Also initializes logging.

fauxmo.cli.**cli**()
Parse command line options, provide entry point for console scripts

## fauxmo.fauxmo module

fauxmo.py

Emulates a Belkin Wemo for interaction with an Amazon Echo. See README.md at <https://github.com/n8henrie/fauxmo>.

fauxmo.fauxmo.**main**(*config_path=None*, *verbosity=20*)
Runs the main fauxmo process

Spawns a UDP server to handle the Echo's UPnP / SSDP device discovery process as well as multiple TCP servers to respond to the Echo's device setup requests and handle its process for turning devices on and off.

**Kwargs:**

**config_path (str): Path to config file. If not given will search for** *config.json* in cwd, *~/.fauxmo/*, and */etc/fauxmo/*.

verbosity (int): Logging verbosity, defaults to 20

## fauxmo.protocols module

protocols.py

Holds asyncio protocols required classes for the Echo's UPnP / SSDP device discovery.

**class** fauxmo.protocols.**Fauxmo**(*name*, *action_handler*)
Bases: asyncio.protocols.Protocol

Mimics a WeMo switch on the network.

Aysncio protocol intended for use with BaseEventLoop.create_server.

**connection_made**(*transport*)

**data_received**(*data*)
Decode data and determine if it is a setup or action request

**handle_action**(*msg*)
Execute *on* or *off* method of *action_handler*

**Args:** msg (str): Body of the Echo's HTTP request to trigger an action

**handle_setup**()
Create a response to the Echo's setup request

**class** fauxmo.protocols.**SSDPServer**(*devices=None*)
Bases: asyncio.protocols.DatagramProtocol

Responds to the Echo's SSDP / UPnP requests

**add_device**(*name*, *ip_address*, *port*)

**connection_lost**(*exc*)

**connection_made**(*transport*)

**datagram_received**(*data*, *addr*)
Check incoming UDP data for requests for Wemo devices

**respond_to_search**(*addr*)
Build and send an appropriate response to an SSDP search request.

## fauxmo.utils module

utils.py

Utility functions for Fauxmo.

fauxmo.utils.**get_local_ip**(*ip_address*)
Attempt to get the local network-connected IP address

fauxmo.utils.**make_serial**(*name*)
Create a persistent UUID from the device name

Returns a suitable UUID derived from *name*. Should remain static for a given name.

**Args:** name (str): Friendly device name (e.g. "living room light")

## Module contents

CHAPTER 2

# Credits

## Development Lead

- Maker Musings https://github.com/makermusings

## Contributors

- Modifications by Nathan Henrie nate@n8henrie.com

CHAPTER 3

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## Types of Contributions

### Report Bugs

Report bugs at https://github.com/n8henrie/fauxmo/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

### Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

### Write Documentation

fauxmo could always use more documentation, whether as part of the official fauxmo docs, in docstrings, or even on the web in blog posts, articles, and such.

### Submit Feedback

The best way to send feedback is to file an issue at https://github.com/n8henrie/fauxmo/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## Get Started!

Ready to contribute? Here's how to set up fauxmo for local development.

1. Fork the fauxmo repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/fauxmo.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ cd fauxmo
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install -e .
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 fauxmo tests
$ python3 setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests if I am using tests in the repo.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.md
3. The pull request should work for Python 3.4 and 3.5. If I have included a `.travis.yml` file in the repo, check https://travis-ci.org/n8henrie/fauxmo/pull_requests and make sure that the tests pass for all supported Python versions.

## Tips

To run a subset of tests: `py.test tests/test_your_test.py`

CHAPTER 4

# Changelog

Will not contain minor changes – feel free to look through `git log` for more detail.

## 0.3.3 :: 20160722

- Added compatibility for `rollershutter` to `handlers.hass`
- Changed `handlers.hass` to send values from a dict to make addition of new services easier in the future

## 0.3.2 :: 20160419

- Update SSDPServer to `setsockopt` to permit receiving multicast broadcasts
- `sock` kwarg to `create_datagram_endpoint` no longer necessary, restoring functionality to Python 3.4.0 - 3.4.3 (closes #6)
- `make_udp_sock()` no longer necessary, removed from `fauxmo.utils`
- Tox and Travis configs switched to use Python 3.4.2 instead of 3.4.4 (since 3.4.2 is the latest available in the default Raspbian Jessie repos)

## 0.3.1 :: 20160415

- Don't decode the UDP multicast broadcasts (hopefully fixes #7)
  - They might not be from the Echo and might cause a `UnicodeDecodeError`
  - Just search the bytes instead
- Tests updated for this minor change

## 0.3.0 :: 20160409

- Fauxmo now uses asyncio and requires Python >= 3.4.4
- *Extensive* changes to codebase
- Handler classes renamed for PEP8 (capitalization)
- Moved some general purpose functions to `fauxmo.utils` module
- Both the UDP and TCP servers are now in `fauxmo.protocols`
- Added some rudimentary pytest tests including tox and Travis support
- Updated documentation on several classes

## 0.2.0 :: 20160324

- Add additional HTTP verbs and options to `RestApiHandler` and Indigo sample to config
  - **NB:** Breaking change: `json` config variable now needs to be either `on_json` or `off_json`
- Make `RestApiHandler` DRYer with `functools.partialmethod`
- Add `SO_REUSEPORT` to `upnp.py` to make life easier on OS X

## 0.1.11 :: 20160129

- Consolidate logger to `__init__.py` and import from there in other modules

## 0.1.8 :: 20160129

- Add the ability to manually specify the host IP address for cases when the auto detection isn't working (https://github.com/n8henrie/fauxmo/issues/1)
- Deprecated the `DEBUG` setting in `config.json`. Just use `-vvv` from now on.

## 0.1.6 :: 20160105

- Fix for Linux not returning local IP
  - restored method I had removed from Maker Musings original / pre-fork version not knowing it would introduce a bug where Linux returned 127.0.1.1 as local IP address

## 0.1.4 :: 20150104

- Fix default verbosity bug introduced in 1.1.3

## 0.1.0 :: 20151231

- Continue to convert to python3 code
- Pulled in a few PRs by @DoWhileGeek working towards python3 compatibility and improved devices naming with dictionary
- Renamed a fair number of classes
- Added kwargs to several class and function calls for clarity
- Renamed several variables for clarity
- Got rid of a few empty methods
- Import devices from `config.json` and include a sample
- Support `POST`, headers, and json data in the RestApiHandler
- Change old debug function to use logging module
- Got rid of some unused dependencies
- Moved license (MIT) info to LICENSE
- Added argparse for future console scripts entry point
- Added Home Assistant API handler class
- Use "string".format() instead of percent
- Lots of other minor refactoring

CHAPTER 5

# Fauxmo README

Python 3 module that emulates Belkin WeMo devices for use with the Amazon Echo.

- Documentation: fauxmo.readthedocs.org

## Introduction

The Amazon Echo is able to control certain types of home automation devices by voice. Fauxmo provides emulated Belkin Wemo devices that the Echo can turn on and off by voice, locally, and with minimal lag time. Currently these Fauxmo devices can be configured to make requests to an HTTP server or to a Home Assistant instance via its Python API and only require a JSON config file for setup.

As of version 0.3.0, Fauxmo uses the new asyncio module and therefore requires Python >= 3.4*. Python >= 3.5 is encouraged, in case I decide to use the new `async` and `await` keywords in the future.

* Fauxmo 0.3.0 required Python >= 3.4.4, but Fauxmo 0.3.2 has restored compatibility with Python >= 3.4.0.

## Usage

### Simple install: From PyPI

1. `python3 -m pip install fauxmo`
2. Make a `config.json` based on `config-sample.json`
3. `fauxmo -c config.json [-v]`

### Simple install of dev branch (from GitHub)

1. `pip install [-e] git+https://github.com/n8henrie/fauxmo.git@dev`

## Install for development (from GitHub)

1. `git clone https://github.com/n8henrie/fauxmo.git`
2. `cd fauxmo`
3. `python3 -m venv venv`
4. `source venv/bin/activate`
5. `pip install -e .`
6. `cp config-sample.json config.json`
7. Edit `config.json`
8. `fauxmo [-v]`

## Set up the Echo

1. Open the Amazon Alexa webapp to the Smart Home page
2. **With Fauxmo running**, click "Discover devices" (or tell Alexa to "find connected devices")
3. Ensure that your Fauxmo devices were discovered and appear with their names in the web interface
4. Test: "Alexa, turn on [the kitchen light]"

## Set fauxmo to run automatically in the background

### systemd (e.g. Raspbian Jessie)

1. Recommended: add an unprivileged user to run Fauxmo: `sudo useradd -r -s /bin/false fauxmo`
   - NB: Fauxmo may require root privileges if you're using ports below 1024
2. `sudo cp extras/fauxmo.service /etc/systemd/system/fauxmo.service`
3. Edit the paths in `/etc/systemd/system/fauxmo.service`
4. `sudo systemctl enable fauxmo.service`
5. `sudo systemctl start fauxmo.service`

### launchd (OS X)

1. `cp extras/com.n8henrie.fauxmo.plist ~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
2. Edit the paths in `~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
   - You can remove the `StandardOutPath` and `StandardErrorPath` sections if desired
3. `launchctl load ~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
4. `launchctl start com.n8henrie.fauxmo`

## Handlers

Fauxmo has an example REST handler class that reacts to on and off commands using the python-requests library as well as a handler for the Home Assistant Python API; these are examples of a multitude of ways that you could have the Echo trigger an action. In `config-sample.json`, you'll see examples of:

- A `GET` request to a local server
- A `POST` request to the Home Assistant REST API
- A `PUT` request to an Indigo server
- Requests to Home Asssistant's Python API

## Configuration

I recommend that you copy and modify `config-sample.json`. Fauxmo will use whatever config file you specify with `-c` or will search for `config.json` in the current directory, `~/.fauxmo/`, and `/etc/fauxmo/` (in that order).

- `FAUXMO`: General Fauxmo settings
  - `ip_address`: Manually set the server's IP address. Optional. Recommended value: `auto`
- `DEVICES`: List of devices that will employ `RESTAPIHandler`
  - `port`: Port that Echo will use connect to device, should be different for each device
  - `handler`: Dictionary for `RESTAPIHandler` configuration
    - `on_cmd`: URL that should be requested to turn device on
    - `off_cmd`: URL that should be requested to turn device off
    - `method`: GET, POST, PUT, etc.
    - `headers`: Optional dict for extra headers
    - `on_json` / `off_json`: Optional dict of JSON data
    - `on_data` / `off_data`: Optional POST data
    - `auth_type`: `basic` or `digest` authentication, optional
    - `user` / `password`: for use with `auth_type`, also optional
  - `description`: What you want to call the device (how to activate by Echo)
- `HOMEASSISTANT`: Section for Home Assistant Python API
  - `enable`: Disable this section by omitting or setting to `false`
  - `host`: IP of host running Hass
  - `port`: Port for Hass access (default: 8123)
  - `password`: Hass API password
  - `DEVICES`: List of devices that will employ `HassApiHandler`
    - `description`: What you want to call the device (how to activate by Echo)
    - `port`: Port that Echo will use connect to device, should be different for each device

    * `entity_id`: Hass identifier used in API, one easy way to find is to curl and grep the REST API, eg `curl http://IP_ADDRESS/api/bootstrap | grep entity_id`

## Troubleshooting / FAQ

- How can I increase my logging verbosity?
  - `-v[vv]`
- How can I ensure my config is valid JSON?
  - `python -m json.tool < config.json`
  - Use `jsonlint` or one of numerous online tools
- How can I install an older / specific version of Fauxmo?
  - Install from a tag:
    * `pip install git+git://github.com/n8henrie/fauxmo.git@v0.1.11`
  - Install from a specific commit:
    * `pip install git+git://github.com/n8henrie/fauxmo.git@d877c513ad45cbbbd77b1b83e7a2f03bf0004856`
- Where can I get more information on how the Echo interacts with devices like Fauxmo?
  - Check out `protocol_notes.md`

### Installing Python 3.5 with pyenv

```
sudo install -o $(whoami) -g $(whoami) -d /opt/pyenv
git clone https://github.com/yyuu/pyenv /opt/pyenv
echo 'export PYENV_ROOT="/opt/pyenv"' >> ~/.bashrc
echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(pyenv init -)"' >> ~/.bashrc
source ~/.bashrc
pyenv install 3.5.1
```

You can then install Fauxmo into Python 3.5 in a few ways, including:

```
# Install with pip
"$(pyenv root)"/versions/3.5.1/bin/python3.5 -m pip install fauxmo

# Run with included console script
fauxmo -c /path/to/config.json -vvv

# Show full path to fauxmo console script
pyenv which fauxmo

# I recommend using the full path for use in start scripts (e.g. systemd, cron)
"$(pyenv root)"/versions/3.5.1/bin/fauxmo -c /path/to/config.json -vvv

# Alternatively, this also works (after `pip install`)
"$(pyenv root)"/versions/3.5.1/bin/python3.5 -m fauxmo.cli -c config.json -vvv
```

## Acknowledgements / Reading List

- Tremendous thanks to @makermusings for the original version of Fauxmo!
  - Also thanks to @DoWhileGeek for commits towards Python 3 compatibility
- http://www.makermusings.com/2015/07/13/amazon-echo-and-home-automation
- http://www.makermusings.com/2015/07/18/virtual-wemo-code-for-amazon-echo
- http://hackaday.com/2015/07/16/how-to-make-amazon-echo-control-fake-wemo-devices
- https://developer.amazon.com/appsandservices/solutions/alexa/alexa-skills-kit
- https://en.wikipedia.org/wiki/Universal_Plug_and_Play
- http://www.makermusings.com/2015/07/19/home-automation-with-amazon-echo-apps-part-1
- http://www.makermusings.com/2015/08/22/home-automation-with-amazon-echo-apps-part-2

CHAPTER 6

# protocol_notes.md

Details on the Echo's interaction with Fauxmo, and how to examine it for debugging.

Tons of information gathered by @makermusings, I *strongly* recommend you start by reading these:

- https://github.com/makermusings/fauxmo/blob/master/protocol_notes.txt
- http://www.makermusings.com/2015/07/13/amazon-echo-and-home-automation

In summary:

1. User tells Echo to "find connected devices" or clicks corresponding button in webapp
2. Echo broadcasts "device search" to `239.255.255.250:1900` (UDP)
3. Fauxmo response includes `LOCATION` of `setup.xml` endpoint for each "device" in config (UDP)
4. Echo requests `setup.xml` endpoint at above `LOCATION` (HTTP) for each device
5. Fauxmo responds with setup information for each device (HTTP)
6. Alexa verbally announces any discovered devices (*really* wish I could mute this – set volume to 1 beforehand if I'll be doing it a bunch), and they also show up in the webapp

Once you understand the basic model of interaction, the next step in debugging is to inspect the actual requests and responses.

The following commands require some tools you might not have by default; you can get them with: `sudo apt-get install tcpdump tshark nmap`. Doesn't matter what you choose regarding the wireshark question you'll get during installation; just read the warning and make a good decision. On OSX, use homebrew to install the same.

First, get the IP address of your Echo. If you don't know it:

```
# Assuming your local subnet is 192.168.27.*
sudo nmap -sP 192.168.27.1/24 | grep -i -B 2 F0:27:2D
```

You should get `Nmap scan report for 192.168.27.XXX` – your Echo IP address. For the examples below, I'll use `192.168.27.100` as the Echo IP address, and `192.168.27.31` as the Pi's IP address (31 as in 3.14, easier to remember).

Next, we'll check out the info being sent to and from the Echo and Fauxmo. In one window, run Fauxmo in verbose mode. In a second window, run the commands below, and check their output when you tell the Echo to find connected devices.

To get an overview of what's going on, start with `tshark`:

```
sudo tshark -f "host 192.168.27.100" -Y "udp"

# Alternatively, only show the Echo's SEARCH requests:
sudo tshark -f "host 192.168.27.100" -Y "udp contains SEARCH"

# Only show the Fauxmo responses (note still using the Echo's IP):
sudo tshark -f "host 192.168.27.100" -Y "udp contains LOCATION"
```

Example output for the first command, showing a few sets of SSDP SEARCH sent by the Echo followed by 4 responses by Fauxmo (1 for each device in sample config).

```
Capturing on 'eth0'
  1   0.000000 192.168.27.100 -> 239.255.255.250 SSDP 149 M-SEARCH * HTTP/1.1
  2   0.046414 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  3   0.064351 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  4   0.082011 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  5   0.101093 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  6   0.104016 192.168.27.100 -> 239.255.255.250 SSDP 149 M-SEARCH * HTTP/1.1
  7   0.151414 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  8   0.171049 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  9   0.191602 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
 10   0.199882 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
 11   0.231841 192.168.27.100 -> 239.255.255.250 SSDP 164 M-SEARCH * HTTP/1.1
 12   0.333406 192.168.27.100 -> 239.255.255.250 SSDP 164 M-SEARCH * HTTP/1.1
```

To get a raw look at all the info, use `tcpdump`. I've cleaned up a bunch of garbage in the below output, but you should still be able to recognize each of the critical components.

```
sudo tcpdump -s 0 -i eth0 -A host 192.168.27.100
```

This should show a ton of detailed info, including all responses sent to / from the Echo. Replace `etho` with your network interface (check with `ip link`) and `192.168.27.100` with your Echo's IP address.

The output should start with several of the Echo's UDP based discovery requests, where you can recognize the `UDP` protocol being sent from the Echo `192.168.27.100` to the network's multicast broadcast `239.255.255.250.1900`, something like:

```
15:48:39.268125 IP 192.168.27.100.50000 > 239.255.255.250.1900: UDP, length 122
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: 15
ST: urn:Belkin:device:**
```

Below that, you should see Fauxmo's responses, also UDP, one for each device in the config. This response provides the Echo with the `LOCATION` of the device's `setup.xml`.

```
15:48:39.513741 IP 192.168.27.31.1900 > 192.168.27.100.50000: UDP, length 386
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=86400
DATE: Sun, 24 Apr 2016 21:48:39 GMT
EXT:
```

```
LOCATION: http://192.168.27.31:12340/setup.xml
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: c66d1ad0-707e-495e-a21a-1d640eed4547
SERVER: Unspecified, UPnP/1.0, Unspecified
ST: urn:Belkin:device:**
USN: uuid:Socket-1_0-2d4ac336-8683-3660-992a-d056b5382a8d::urn:Belkin:device:**
```

Somewhere below that, you'll see the Echo request each device's `setup.xml` (based on the `LOCATION` from the prior step), this time TCP instead of UDP.

```
15:48:39.761878 IP 192.168.27.100.39720 > 192.168.27.31.12341: Flags [P.], seq 1:68,␣
↪ack 1, win 274, options [nop,nop,TS val 619246756 ecr 140303456], length 67
GET /setup.xml HTTP/1.1
Host: 192.168.27.31:12341
Accept: */*
```

And somewhere below that, Fauxmo's setup response, for each device in the config, also TCP:

```
15:48:39.808164 IP 192.168.27.31.12342 > 192.168.27.100.59999: Flags [P.], seq 1:608,␣
↪ack 68, win 453, options [nop,nop,TS val 140303462 ecr 619246754], length 607
HTTP/1.1 200 OK
CONTENT-LENGTH: 375
CONTENT-TYPE: text/xml
DATE: Sun, 24 Apr 2016 21:48:39 GMT
LAST-MODIFIED: Sat, 01 Jan 2000 00:01:15 GMT
SERVER: Unspecified, UPnP/1.0, Unspecified
X-User-Agent: Fauxmo
CONNECTION: close

<?xml version="1.0"?>
<root>
<device>
<deviceType>urn:Fauxmo:device:controllee:1</deviceType>
<friendlyName>fake hass switch by REST API</friendlyName>
<manufacturer>Belkin International Inc.</manufacturer>
<modelName>Emulated Socket</modelName>
<modelNumber>3.1415</modelNumber>
<UDN>uuid:Socket-1_0-cbc4bc63-e0e2-3a78-8a9f-f0ff7e419b79</UDN>
</device>
</root>
```

Then, to get a *really* close look at a request, well go back to `tshark`. For example, we can add the `-V` flag to get a **ton** more info, and add `-c 1` (count) to limit to capturing a single packet, and further refine the capture filter by specifying that we only want to look at packets sent **from** the Pi **to** the Echo.

```
sudo tshark -f "src 192.168.27.31 and dst 192.168.27.100" -c 1 -V
```

At the bottom, you should find the `Hypertext Transfer Protocol` section contains the same `setup.xml` response we found in the `tcpdump` output above.

You can also send requests from another device on the network to check out Fauxmo's responses and ensure that they're getting through the network. For example, to simulate the Echo's device search, run the following from another device on the network, in two different windows:

```
# Seems to work with `nc.traditional` on Raspberry Pi, not yet working for me on OSX
# Window 1: Listen for response on port 12345 (should show up once second command is␣
↪sent)
nc.traditional -l -u -p 12345
```

```
# Window 2: Send simulated UDP broadcast device search (from port 12345)
echo -e '"ssdp:discover"urn:Belkin:device:**' | nc.traditional -b -u -p 12345 239.255.
↪255.250 1900
```

To request a device's `setup.xml`, using the device's `port` from `config.json`:

```
# Send a request for the `setup.xml` of a device from the sample config
curl -v 192.168.27.31:12340/setup.xml
```

The above commands may seem a little complicated if you're unfamiliar, but they're immensely powerful and indispensable for debugging these tricky network issues. If you're not already familiar with them, learning the basics will serve you well in your IoT endeavors!

To verify that Fauxmo is working properly, check for a few things:

1. Is the Pi consistently seeing the Echo's `M-SEARCH` requests?
2. Is Fauxmo consistently replying with the `LOCATION` responses?
3. Is the Echo then requesting the `setup.xml` (for each device)?
4. Is Fauxmo consistently replying with the setup info?

If you can confirm that things seem to be working through number 4, then it would seem that Fauxmo is working properly, and the issue would seem to be elsewhere.

## On and Off Commands

One way to examine exactly what the Echo sends to one of your connected Fauxmo devices (i.e. one that *already* works as expected) is to first **stop** Fauxmo (to free up the port), then use netcat to listen to that port while you trigger the command. E.g. for a Fauxmo device configured to use port `12345`, run `nc.traditional -l 12345` and then tell the Echo to "turn on [device name]". The Echo will notify you that the command failed, obviously, because Fauxmo isn't running, but you should be able to see exactly what the Echo sent.

These are the requests that the Echo sends to Fauxmo when you ask it to turn a device...

### On

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"
Content-Length: 299

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>1</BinaryState>
</u:SetBinaryState>
</s:Body>
</s:Envelope>
```

### Off

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"
Content-Length: 299

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>0</BinaryState>
</u:SetBinaryState>
</s:Body>
</s:Envelope>
```

Several similar terms can be used instead of `On` and `Off`, e.g. `Open` and `Close`; the response looks identical. This Reddit post has a good number more that work. NB: the `Dim` commands in the post don't seem to work (likely incompatible with Wemo devices, so the Echo doesn't even try to send them).

CHAPTER 7

# Indices and tables

- genindex
- modindex

# Python Module Index

## f

# Index